

Documentation for dynsys.h and dynsys.c

Steven Andrews, © 2004

See the document “LibDoc” for general information about this and other libraries.

```
typedef struct phpt {
    int sp;
    float *sa;
    int fp;
    int fs;
    float *fa; } *phptr;

#define MaxIntPar 10

int ODEuler(void (*eqm)(float *,float *,float *),float *k,int p,float *u,float
    tf,float dt,int (*trackfn)(float,int,float *));
int ODErk4(void (*eqm)(float *,float *,float *),float *k,int p,float *u,float
    tf,float dt,int (*trackfn)(float,int,float *));
int ODErkas(void (*eqm)(float *,float *,float *),float *k,int p,float *u,float
    tf,float dt,int (*trackfn)(float,int,float *));
void EQMsho(float *u,float *k,float *dudt);
void EQMlorenz(float *u,float *k,float *dudt);

phptr phptalloc(int sp,int fp,int fs);
void phptfree(phptr u);
int phptsave(phptr u,char *fnames,char *fnamef);
phptr phpload(char *fnames,char *fnamef);
int ODEFuler(void (*eqm)(phptr,void *,phptr),void *k,phptr u,float *Dt,float
    intpar[],int (*trackfn)(float,phptr,void *),void *trackptr);
int ODEFrk4(void (*eqm)(phptr,void *,phptr),void *k,phptr u,float *Dt,float
    intpar[],int (*trackfn)(float,phptr,void *),void *trackptr);
int ODEFrkas(void (*eqm)(phptr,void *,phptr),void *k,phptr u,float *Dt,float
    intpar[],int (*trackfn)(float,phptr,void *),void *trackptr);
void EQMFzero(phptr u,void *k,phptr dudt);
void EQMFsho(phptr u,void *k,phptr dudt);
void EQMFdiff(phptr u,void *k,phptr dudt);
void EQMFwave(phptr u,void *k,phptr dudt);
int TKFticker(float t,phptr u,void *tkptr);
int TKFtimeplot(float t,phptr u,void *tkptr);
int TKFshowfield(float t,phptr u,void *tkptr);

/* start of 2004 version */

typedef struct odestruct {
    int dim;
    int order;
    float *dtptr;
    void *systemptr;
    int (*eqm)(void *);
    float **state0;
    float **state1;
    float *k1;
    float *k2; } *odeptr;
```

```

odeptr allocodestruct(int dim,int order,float *dtptr,void *systemptr,int
    (*eqm)(void *));
void freeodestruct(odeptr ode);
int runodestruct(odeptr ode);
int odestructexample(void);

/* end of 2004 version */

```

First parts written 1/29/97. Field routines added 6/97; some testing. Slight additions 1/00. Reformeted documentation and added a new independent section 6/04, called '2004 version'.

This library is really three parallel libraries, each of which integrates ordinary differential equations. Routines without an "F" are designed to work together, and are for small dimensional phase space (presently set for Pmax=10 dimensions, but easily enlarged). Routines with an "F", which stands for "field", are designed for high dimensional phase space, including fields. The latter routines are also a little more general and a little more careful about being proper and checking for errors. The data structure `struct phpt` is only used in the "F" routines. Routines that start with ODE are differential equation integrators, ones that start with EQM are some useful equations of motion, and ones starting with TK are for tracking the results of the integrations. Routines in the section called '2004 version' are different yet, and use a data structure called `odestruct`, pointed to by `odeptr`. The 2004 routines compile very slowly when maximum compiler optimizations are used.

The notation for the first two sections is largely taken from Stuart and Humphries's *Dynamical Systems and Numerical Analysis* while the algorithms are from *Numerical Recipes in C*. The code is similar to that in *Numerical Recipes*, but mine are a little faster and seem to me to be simpler.

The "F" routines use the structure `struct phpt`, pointed to by the type `phptr`, to define a point in phase space. The members `sp`, `fp`, and `fs` are, respectively, the number of scalar dimensions in phase space, the number of field dimensions, and the size of the fields. The members `sa` and `fa` are the actual scalar and field arrays. `sa` is indexed from 0 to `sp-1`, while `fa` is indexed with columns 0 to `fp-1` to identify the field and with rows from 0 to `fs-1` to identify the location in the field. Thus, for example, `u->fa[u->fp*(u->fs-1)+0]` is the last element of the first field of the `phptr u`. Memory for `struct phpts` are allocated with `phptalloc` and freed with `phptfree`. The former routine returns a `phptr`, set up with the `sp`, `fp`, and `fs` members defined (using the input arguments) and space allocated in the appropriate arrays. If memory allocation failed, `phptalloc` returns a 0.

The 2004 routines use a rather different scheme. Rather than integrating an ODE over a long period of time and having the integrator call back to a tracking function, the new routine are designed for only a single time step. The idea is that they are just simply routines that can be called from an existing program rather than a design that a program needs to be built around. `odestruct`, pointed to by a `odeptr`, is a structure that contains all the useful information about an ODE, including scratch space for the integrator. `dim` is the number of dimensions of the system of equations, which is the number of differential equations. `order` is the order of the Runge-Kutta integrator, which is allowed to equal 1 (for Euler integration), 2 (for mid-point method), or 4 (for 4th order Runge-Kutta). When adaptive step sizing is written, `order` will also be allowed to equal 5. `dtptr` is a pointer to the variable that contains the time step. `systemptr` is a pointer to a structure that contains everything that is known about the system, cast as a `void*`; it is never used in any of the routines here, but it is passed on to the relevent function with the

equations of motion. `eqm` is the function for the equations of motion, where its only parameter is the system, pointed to by `systemptr`. `state0` and `state1` are `dim` long lists of pointers, where each pointer points to one of the time dependent variables. When the integrator is done with one time step, the variables pointed to by `state0` contain the state of the system at the end of the time step, while those pointed to by `state1` contain the state of the system at the beginning of the time step. While the integrator is busy, `state1` points to the current system values and `state0` is supposed to be returned to the integrator from the equations of motion function with the time derivative for each variable. This is made clearer below. `k1` and `k2` are each `dim` long scratch space vectors for the integrator. They are only allocated as needed. If the `odestruct` is allocated with one `order` value, it is allowable to change the `order` to a lower value, but the `order` may not be raised above the original value because the required memory will not have been allocated.

Pre-2004 routines

ODEeuler and ODEFeuler

These use Euler's method of differential equation integration, which is simple but of low quality. The non-"F" ODE routines all take in identical arguments. Going through them in order, pass in a function name for the equations of motion, a vector of constant parameters (not used by the integrator, but passed on to `*eqm`), the dimension of phase space, the initial conditions (`u[0..p-1]`), the final time, the time step, and the name of a function to track the results. The `*eqm` function is sent a point in phase space, the constant vector `k`, and an uninitialized array in which the first time derivatives are to be returned. Note that the time is not sent, so if it is needed, it needs to be added as another dimension. If the `*eqm` function returns a non-zero value, the integration is stopped. If you don't want to watch or record the results as they are produced, making `*trackfn` a NULL pointer will tell the integrator to keep going. Otherwise, `*trackfn` is sent the time, the dimension of phase space, and the present point in phase space. It is called after the first time step, and every step thereafter up to, but not including `tf`. The tracking function should return zero for stable operation and a non-zero integer to abort the integration.

The arguments to the "F" integrators are similar to those for the other integrators. Going through them in order, pass in a function name for the equations of motion, a pointer to the constant parameters for the equations of motion, the starting point in phase space, a pointer to the total time increment, a set of parameters for the integrator, the name of a function to track results, and a pointer to any information needed by the tracking function. As above, the `*eqm` function is sent a point in phase space, the pointer to the constant parameters, and an uninitialized `phptr` in which the gradient is to be returned. The constant parameters for the equations of motion, pointed to by `k`, may be an array of numbers, a `struct phpt`, or whatever other data type is expected by the equations of motion. The parameters to the integrator could be numerous in principal, but in fact all of the routines written so far only look at the first element, `intpar[0]`, to get the integration step size (or the initial step size for `ODEFrkas`). The constant `MaxIntPar` sets the maximum required size of the `intpar` array. Making `*trackfn` a NULL pointer tells the integrators to continue until they finish. Otherwise, they call `*trackfn` after each step, with the time, the present point in phase space, and a pointer to any information for the tracking function. Again, this last pointer is completely general. The tracking function is called after one time step, and then each time step thereafter, up to, but not including, `Dt`. If an integrator is interrupted before it finishes, it returns the value 1; otherwise it returns 0. Regardless of how it terminates, the actual time integrated is returned as `*Dt` and the phase space point for that time is returned in `u`.

ODErk4 and ODEFrk4

These use a fixed step fourth order Runge-Kutta algorithm, which is much better than Euler's method. See above for arguments.

ODErkas and ODEFrkas

These use a fifth order Runge-Kutta algorithm with adaptive step sizing. Thus, the routines take large steps in smooth areas of phase space and small steps in more difficult regions. Step sizing is supposed to work such that the error on each step is as high as possible, but no more than that taken on the first step, for which the step size is supplied by the user. See above for arguments.

EQMsho

This is a set of equations of motion for a simple harmonic oscillator. It requires one constant, ω^2 (store this in `k[0]`). The first phase space dimension is the position, while the second is the velocity. The equations are: $dx/dt = v$; $dv/dt = -\omega^2 x$.

EQMlorenz

This contains the Lorenz equations. The phase space dimensions are x , y , and z , while the constant parameters are σ , r , and b . Equations are: $dx/dt = \sigma(y-x)$; $dy/dt = rx - y - xz$; $dz/dt = xy - bz$.

phptsave and phptload

Phase points may be saved to or loaded from disk with the routines `phptsave` and `phptload`. For both routines, the file names may be sent with the `fnames` and `fnamef` arguments, or, if they are set to `NULL`, the routines ask the user for file names. The `fnames` file is a file of the scalar array and `fnamef` file is a matrix of the field elements. `phptsave` returns 1 if saving was successful, and 0 otherwise.

EQMFzero

This requires no constant parameters and returns a zero gradient everywhere.

EQMFsho

This contains the equations of motion for a simple harmonic oscillator, inputting ω^2 in `k[0]` (cast to a `float`).

EQMFdiff

This contains the equations of motion for diffusion in the first field (`u->fa[0][j]`) with diffusion constant `k[0]`.

EQMFwave

This contains the wave equation in the first two fields with c^2 from `k[0]`.

TKFticker

This displays the first two scalar parameters as text.

TKFtimeplot

This plots the first scalar parameter as a function of time.

TKFshowfield

This plots the first field, replacing it at each time step.

2004 routines

allocodestruct

This allocates an odestruct and returns an odeptr that points to it, assuming allocation was successful, or NULL if not. dim is the number of differential equations (number of time dependent variables), order is the maximum order of integration that will be used (1, 2, 4, or 5), dtptr is a pointer to the variable that contains the time step, systemptr is a pointer to a structure that contains all the relevant information for the equations of motion function cast as a void*, and eqm is a pointer to the function with the equations of motion. See below for use advice. This function allocates all memory that is required and assigns all structure elements of the odestruct, except for the pointers in the state0 and state1 lists. These are left as all NULLs and need to be set elsewhere.

freeodestruct

This frees an odestruct, including all memory that was allocated using allocodestruct. It does not free memory that was allocated elsewhere.

runodestruct

This performs the integration of an odestruct over one time step. It uses Euler, Mid-point, fixed step Runge-Kutta, or adaptive step Runge-Kutta, as appropriate. On input, state0 pointers point to the current state of the system and state1 pointers are ignored. On output, state1 pointers point to the past state of the system and state0 pointers point to the new state. Note that the derivative information is not set to zero before the equations of motion function is called. Orders 1, 2, and 4 seem to be quite reliable and robust, although order 5 does not always work well. It needs to be checked out more thoroughly.

odestructexample

This is an example of the use of the 2004 routines using a simple harmonic oscillator. Note that the oscillator amplitude increases exponentially if the integrator order is set to 1, whereas it is nearly constant with an order of 2 and it is quite good with an order of 4. The equations of motion are: $dx/dt = v$; $dv/dt = -\omega^2 x - 2\gamma v$. γ is a damping term. According to basic physics, the exact theoretical result is: $x = Ae^{-\gamma t} \cos(\omega_1 t - \phi)$ where $\omega_1 = (\omega^2 - \gamma^2)^{1/2}$.